IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application for United States Patent:

SERVER-SIDE DIGITAL SIGNATURE SYSTEM

1

Background of the Invention

2

3

4 1. Field of the Invention

5 The present invention relates generally to methods and systems for authenticating

6 data. More specifically, the invention relates to a method, apparatus and product for

7 providing digital signatures on electronic documents and for authenticating the

8 documents by verifying their signatures.

9

10 2. Background of the Invention

11 It is currently possible to create legally-binding documents using handwritten

12 signatures and by notarizing the handwritten signature, if required. A handwritten

13 signature provides a level of assurance that the written document was executed by person

14 identified by the signature, and prevents repudiation of the signed instrument by the

15 signer.

16 Computer-based methods of producing documents are becoming more prevalent.

17 Electronic documents are replacing written contracts, orders, payment instruments,

18 account statements, invoices, and other documents that have historically been signed by a

19 written signature. It is frequently advantageous to have a document that has been

20 produced and is being stored in digital form to have a digital signature applied to it so

21 that the authentication of the signer can later be verified. The digitally-signed electronic

22 document can then be transmitted for processing, without the need for a signed paper

23 instrument.

1   A need has arisen for alternative mechanisms for creating and authenticating

2   legally binding electronic documents and communications. Digital encryption, digital

3   message digests, digital signatures, and digital certificates are some of the existing

4   cryptographic tools that are used in the present invention to address this need.

5   Public key cryptography is used in some systems to provide such a digital

6   authentication. Public key cryptography uses a two-key pair, typically referred to as a

7   public key and a private key. These two keys appear to be completely independent, yet

8   share an important property; data encrypted by the first can be recovered by the second,

9   and vice versa. RSA is the most well-known public-key algorithm.

10   If one of the numbers is kept private and the other key is made available, then

11   anyone can use the public key and encrypt a message so that only the intended recipient

12   can read it. By encrypting with the private key, anyone can verify with the public key and

13   be assured that only the private key-holder performed a specific operation, thus providing

14   a non-repudiation function.

15   Public-key encryption is computationally expensive. However, when public-key

16   encryption is combined with hashing, a powerful digital signature becomes possible.

17   Hashing involves taking an arbitrary stream of bytes and processing it down to a small

18   number of bytes called the hash. The processing is such that no two streams will result in

19   the same hash. There are a number of hashing algorithms, including MD5, SHA-1, SHA-

20   256, and RIPE-MD160. A digital signature can be produced by the following algorithm:

21   1. Take an arbitrary set of data of any size;

22   2. Hash the data;

23   3. Encrypt the hash with X's private key; this is X's signature on the data.

1   To verify the signature:

2   1. Take the same set of data;

3   2. Hash the data using the same algorithm;

4   3. Decrypt the sender's hash with their public key;

5   4. If the two hashes are the same, the signature is valid.

6

7   An XML digital signature object is currently provided in XML systems. Bytes of

8   data to be signed are collected into a digital document, and a digital signature is created

9   over the data content. The data content is transformed before it is signed. A "reference"

10   is created that names the content, identifies data transformation and hashing algorithms,

11   and includes the hash. To produce the XML-digital signature, the references are hashed,

12   and this hash is signed.

13   The XML data structure for digital signatures contains the following elements:

14   Signature

15   SignedInfo -- reference to the content being signed

16   SignatureValue -- the actual signature, which is a signed hash of the SignedInfo

17   data block

18   KeyInfo -- identifies the signer and his key

19   SignedInfo Reference

20   Name of the content

21   Name of data transformation to be performed

22   Type of hashing to perform

23   Hash Value

1   According to current XML data structures, the *Signature* element has a *SignedInfo*

2   element within it. The hash of the *SignedInfo* element is what is actually signed. The

3   *SignedInfo* element includes one or more *Reference* elements. Each *Reference* element

4   contains the hash of the content, the URI and information on the hashing algorithm that

5   was used for the hash. The XML digital signature recommendation processing rules

6   require that in order to verify a signature, each Reference hash must be validated, and the

7   signature over the *SignedInfo* element must be validated.

8   Using current digital signature systems, a client computer is connected over a

9   network to a server computer. The client must generate a signature using cryptographic

10   hardware and/or software. The client conveys the signature and optionally its public key

11   to the intended recipient. Using current systems, the client system must have much of the

12   functionality of the digital signature system, and the client must generate the encryption

13   keys.

14   It is therefore an object of the present invention to provide a digital signature

15   system that has all of its functionality on a server, including generation and maintenance

16   of client keys.

17   It is a further object of the present invention to provide a digital signature system

18   that allows a remote client to send a data object to the server, and the server generates a

19   digital signature on the data object and returns the signature to the client.

20   It is an additional object of the invention to provide a digital signature system that

21   verifies signatures that it previously produced on a data object upon request from a client.

6

1   It is also an object of the present invention to allow the content being signed to be

2   "split" across the two parties, so that the compromise of one does not result in the

3   generation of unintended signatures.

4

5

Summary of the Invention

These and other objects are attained by the invention, one aspect of which comprises a signing server that creates a signature for a data object, associates the signature and the data object with a signed object, and later authenticates the signed object. According to the invention, the client computer accesses the signing server, such as by using a browser, and then transmits the data object from the client to the server via a communications channel such as the Internet. Upon receiving the object, the server processes the object to generate a signature, associates the signature with the object to generate a signed object, and transmits the signed object back to the client.

Subsequently, upon a request from the client, the server authenticates the signed object by deriving the original data object and the signature from information obtained from the signed object sent by the client. The server then generates a comparison value by hashing the original data object to produce a second hash, and comparing the hash value in the signature to the second hash. The server also checks the hash value in the signature using the user's public key. If the document is authenticated, the server notifies the client that the authentication was successful.

According to the invention, the server generates and manages the clients' keys that are used to generate a signature. In the preferred embodiment, the server authenticates the client upon request and then assigns a private key to the client. The client can then send a data object to the server to be signed. The server creates a signature for a specific client by performing a hash function on the data object to produce

8

1   a hash total, and then performing an encyphering process on the hash total using the

2   client's private key.

3   A signed object can have a detached signature, an enveloped signature, an

4   enveloping signature, or a signature with the hash of the data object. When the signed

5   object has a detached signature, the signed object is actually a new, second object, that

6   contains the signature and the address of the signed object. When the signed object has

7   an enveloped signature, a new object is also created that contains the original object and

8   the signature. When the signed object has an enveloping signature, the signature is

9   placed within the original object, which becomes the signed object.

10   According to one embodiment of the invention, when the server creates a signed

11   object, it assigns a signature field and a signature property field to the signed object. The

12   signature field contains the signature that is generated by the server. The signature

13   property field contains a timestamp, a client identifier, key information, or other data that

14   is signed by a key generated and maintained by the server.

15   When the client subsequently requests verification of a signed object, the server

16   obtains the data object and the signature from information in the signed object. The

17   server authenticates the object by verifying the original signature, which is obtained from

18   the signature field of the object. The server generates a second hash using the original

19   data object and the data in the signature property field, and compares the hash from the

20   signature with the second hash. If the hash values match, the server generates a "valid"

21   verification response, which it sends to the client. Alternatively, the server can create a

22   signature on the verification response, creating a signed verification receipt.

1   Many other implementations of the present invention are available

2   according to the present invention, as will be seen by the following description of

3   the invention.

4

Brief Description of the Drawings

Fig. 1 shows the system configuration of the present invention, including a signing server connected to a client over the Internet.

Fig. 2 shows a Signing Request and a Signing Response as implemented in the configuration shown in Fig. 1.

Fig. 3 shows a Verification Request and a Verification Response as implemented in the configuration shown in Fig. 1.

Fig. 4 illustrates how the server generates and assigns private keys to three connected clients.

Fig. 5 shows a method the server uses to pick a private key to generate a signature for an object shown in greater detail.

Fig. 6 illustrates server's generation of a signature for a data object.

Fig. 7 shows how a signature can also be generated using the hash of the object, the private key, and the server key.

Fig. 8A illustrates the generation of a signature for a data object, creating a signed object.

Fig. 8B shows the signed object having a detached signature.

Fig. 8C shows the signed object having an enveloped signature.

Fig. 8D shows the signed object having an enveloping signature.

Fig. 8E shows the signed object having a hash of the data object stored in the signature.

11

1

Detailed Description of the Invention

2

3

4   The present invention is a server-side digital signature system. According to the

5   invention, a client sends a data object, or document, along with a signing request to the

6   server. The server, using the client's key stored on the server, generates a signature for

7   the data object, creates a signed object, and returns the signed object to the client. The

8   client can subsequently send a verification request, along with the signed object, to the

9   server. The server verifies the signature by generating a new hash using the original data

10   and the client's private key, and comparing the new hash to the hash derived from the

11   signature that was assigned to the signed object. All of the functionality of the digital

12   signature system is performed by the server, including creating, storing, controlling and

13   managing the clients' signings keys, other encryption keys and the hash functions used by

14   the system.

15   Various types of clients can access the server-side digital signature system of the

16   present invention. For example, a user-interface based application can allow individuals

17   to sign documents. An enterprise application could use the signing server to sign

18   transactions. Document management or workflow frameworks can call out to the signing

19   server to sign documents. Other basic user applications, including Java-based

20   applications and browser-based user interfaces, as well as interfaces to allow integration

21   with enterprise applications or document management frameworks are possible, as will

22   be apparent to those familiar with the art.

1   Referring to Fig. 1, a client 100 connects to the server 120 over a communication

2   channel such as the Internet 110. The client 100 must first authenticate itself with the

3   signing server 120 of the present invention, and send a user or client ID 130 to the server

4   over the communications channel. According to the present invention, the server requires

5   authentication of all signers and verifiers in order to use its signing services.   Signers

6   and verifiers using the system are registered on the server using the Secure Sockets Layer

7   (SSL). The default authentication is the name/password combination secured by SSL,

8   however, in alternate embodiments, other authentication systems may be used. Client-

9   side certificates are an authentication option. All signing and verification operations are

10   logged at the server.

11   The client can authenticate to the server using one of several methods. The client

12   sends information representing its identity to the server using either a password, an

13   encrypted data channel, a public key-based processing step, or by presenting a client

14   certificate. The client and server can mutually authenticate using a zero-knowledge proof

15   algorithm. After the server has authenticated the identity of the client, the server assigns

16   a private key to the client. The particular signing key to be used by the server is specified

17   using the user/client ID that is passed to the server by the client. In the preferred

18   embodiment, the e-mail address of the user is used to identify the user, and is named

19   within the Client ID 130.

20   Fig. 2 shows a Signing Request and a Signing Response according to the present

21   invention. A client 100 sends a Signing Request and a data object 210 to a server 120

22   over a connection such as the Internet. The server 120 processes the object and generates

23   a signature 225 and assigns it to the object, creating a signed object 230. The server then

13

1   returns a Signing Response and the signed object to the client 100. The Signing

2   Response includes "Success" or "Failure", indicating whether the signature was

3   successfully generated.

4   A signed object can have a detached signature, an enveloped signature, an

5   enveloping signature, or a signature containing the hash of the data object. When the

6   signed object has a detached signature, the server creates a signed object that contains the

7   signature and the address of the data object. When the signed object has an enveloped

8   signature, the server creates a signed object that contains the data object and the

9   signature. When the signed object has an enveloping signature, the server writes the

10   signature within the original data object, which then becomes the signed object. A signed

11   object can also contain the hash of the data object contained within the signature.

12   Fig. 3 shows a Verification Request and a Verification Response according to the

13   present invention. Client 100 can later send the signed object 230 to the server 120 for

14   verification. The server verifies the signature by obtaining the data object 210 and the

15   hash from the first signature 225 from the signed object. The server generates a second

16   hash 310 using the data object and compares the hash from the first signature 225 with

17   the second hash 310. If the signatures match, the signature is valid. The server returns an

18   indicator 320 showing the status of the signature, either valid or invalid.

19   Referring to Fig. 4, client A 405, client B 100 and client X 435 are all clients

20   using the digital signing server 120. The server 120 generates and assigns private keys to

21   all of its clients, and as shown in this example, has generated and maintains private keys

22   410, 420, 430. When client 100 sends a signed object 230 to server 120 for verification,

23   the server 120 determines, based on a specified system policy, available data or a

14

1   specified algorithm, which private key 420 to use to generate a verifying hash. In the

2   preferred embodiment, the particular signing key to be used is specified using the user e-

3   mail name, or Client ID 130 that was shown in Fig. 1. The client ID, or identification

4   element, was passed to the server by the client upon authentication, as shown in Fig. 1.

5   In alternate embodiments, the key may be specified using an issuer name and serial

6   number, a key identifier, or by other techniques. The identification element is discussed

7   in more detail below.

8   Referring to Fig. 5, the method to pick a private key to generate a signature for an

9   object is shown in greater detail. The server uses the client ID 130, and other data 510 to

10   select a private key from the keys 410, 420, 430 that are stored on the server. The server

11   picks a key to use at 520, and generates a signature for object 210 by hashing the object

12   and then encrypting the hash. The signature is stored with the object 210 in a new object,

13   the signed object 230.

14   As shown in Fig. 6, the system generates a signature of a data object 210. Object

15   210 is hashed, according to a pre-determined hash function 620. The client ID is

16   determined by the server, and is used by the server to select a key 420. The server also

17   produces a timestamp, and generates a signature 530 using the hash, the chosen key, and

18   the timestamp. These functions are described in more detail below.

19   As shown in Fig. 7, the signature 530 can also be generated using the hash of the

20   object, the private key, and the server key, and that signature is incorporated into a

21   signature 530 for the client.

22   As shown in Fig. 8A, the system generates a signature 530 for data object 210,

23   creating signed object 230. As shown in Fig. 8B, the signature can be detached. In this

15

1   case, the signed object 230 contains the address of the data object 810 and the signature

2   530. Fig. 8C shows an enveloped signature. Here, the signed object 230 contains the

3   data object 210 itself and the signature 530. Fig. 8D shows an enveloping signature,

4   where the signature 530 is placed within the original data object 210, which becomes the

5   signed object 230. As shown in Fig. 8E, a hash 820 of the data object 210 can be stored

6   in the signature 530, which becomes the signed object 230. A hash of the object is stored

7   in the signature when the client requests that only the signature be returned.

8

9   Description of a Preferred Embodiment

10   In the preferred embodiment of the present invention, the system is implemented

11   using a modification of the standard XML Digital Signature Recommendation, although

12   many other implementations are possible. The modified XML structure will be referred

13   to as a Hancock XML in this specification, and will be used to illustrate a particular

14   embodiment of the invention. Appendices A through S show examples of specific

15   Hancock XML code implementing the preferred embodiment.

16   Client requests to the server and the server's responses to the clients are XML

17   documents. This allows a variety of possible interfaces to the present invention,

18   including a HTML form interface with a CGI backend that produces the XML requests, a

19   client that generates an XML request and transmits the request to the server using the

20   SOAP application protocol over HTTP, and a client interface that reads XML requests

21   from files.

22   According to the present invention, the standard XML *Object* element has an

23   additional element, named the *HancockSignatureProperties* element. In addition, the

16

1   Server-Side digital signature provides signature attributes that are unique to the present

2   invention. The *HancockSignatureProperty* element is always part of the Hancock

3   signature, and ensures the integrity of the Hancock signature properties. The

4   *HancockSignatureProperty* element is always included in the *Signature* element.

5   According to the present invention, requests and responses are XML documents.

6   The system provides Signing Requests, unsigned Verification Responses and signed

7   Verification Receipts. The only signature on a Signing Request and a Verification

8   Response is the requested signature on the document. A signed Verification response is a

9   verification response that has been signed for the relying party, the party who requested

10   the verification.

11   Hancock XML elements are defined within the Hancock namespace. For

12   example, a signing request would be: xmlns="urn:caveosystems:sign1". A verification

13   request would be: xmlns="urn:caveosystems:verify1".

14

Signing Request

16   Appendix A shows an example of a Hancock XML *SigningRequest* element that

17   is sent from a client to the server. According to the preferred embodiment, a Hancock

18   *SigningRequest* is an XML document that includes a document element, a signing policy

19   and an indication of how the result is to be returned. The document element specifies the

20   document to be signed, either by including the data itself or the address of the data. An

21   address is specified by using the URI element. If the document element contains actual

22   data, the data may be base64 encoded. This is specified with the *Algorithm* attribute of

23   the Document element.

17

1   The *SignaturePolicy* element is used to specify how the signing is to be

2   performed. This includes the type of signing, the signing key to use, the required

3   cryptographic strength required for the signature, the formatting of the signature, whether

4   or not to include the *<keyinfo>* element, and a *SSDSSignatureProperties* element. The

5   signature policy also indicates whether the signer can countersign, whether the timestamp

6   should be secure, whether a manifest will be used to include the references to the signed

7   documents. In alternate embodiments, one or more of these parameters may be used,

8   giving a different combination of signing policy attributes possible. For example, the

9   Policy element can also be used to specify document canonicalization and transform

10 requirements.

11   As shown in Appendix A, the client is requesting detached signing and that only

12 the new signature is to be returned in the response. The signing key to use is specified

13 using the *KeyName* element within the *KeyInfo* element. The *KeyNameType* indicates

14 that the key for this client is associated with the e-mail address of the user. The key can

15 also be specified using an *X509Data* element, by defined the XML Digital Signature

16 recommendation. The key can also be specified using an issuer name and serial number,

17 a key identifier, or by other techniques.

18   The *SigningRequest* allows the client to specify how much of the signer's

19 certificate chain to include. In this case, "All" indicates to use all of the certificate chain.

20 The *Type* attribute is optional and specifies how to represent certificates in the certificate

21 chain. Defining the *Type* attribute as "Certificate" indicates that the actual certificate is

22 to be used, and is base64 encoded. An alternative, such as certificate hashes or identifiers

23 may be specified. All certificates in the chain are represented in the same manner.

18

1   The *SigningRequest* specifies how the response is to be returned using the *Return*

2   attribute. In Appendix A, Return="Signature", indicates that only the signature is to be

3   returned in the signed object. The client could also request that the entire data object or

4   the URI address of the data object be returned in the signed object.

5   In the code shown in Appendix A, the *Document URI* is specified as the url of the

6   data object. The client is therefore specifying which data object to sign using the address

7   of the data object document. The *Document* Element may specify more than one

8   document to be signed if a signature is to be produced over several documents.

9   Appendix B is example code showing a *SigningRequest* where the actual

10   document is included in the request. The user has specified receiving an enveloped

11   signature. The private key to be used is specified by the issuer and the serial number of

12   the associated public key certificate. No Return attribute is specified, so the default

13   *Return="Signature"* will be used.

14   Appendix C shows a sample purchase order document to be signed. This

15   document is produced at the client, and the client will send it to the server to be signed by

16   including it in a *SigningRequest*.

17   Appendix D shows the purchase order of Appendix C included in a

18   *SigningRequest*. The data to be signed is included in the body of the request, and the

19   signature is to be enveloped. The key to be used is specified using the user's e-mail

20   address.

21   An end-user can define user-specific properties using the *UserProperties* element

22   in the *SigningRequest*. The *UserProperties* element is inside an *Object* element in the

19

1   *Signature* element, and a reference to the *UserProperties* is included in *SignedInfo*. This

2   is discussed in more detail below.

3

4   Signing Response

5   Referring again to Fig. 2, when a signing request is sent to the server, the server

6   creates a *SigningResponse* element. The *SigningResponse* element includes a *Status*

7   element having a Type attribute of "Success" or "Failure", and additional information

8   regarding the status may be sent within the *SigningResponse* element.

9   Appendix E shows code representing the enveloped signed object that results

10 from generating a signature over the *SigningRequest* of Appendix D. The data itself is

11 returned in the signed object, as an enveloped signature was requested.

12   Appendix F shows the code representing the enveloped signed object of Appendix

13 E in a *SigningResponse* element. The data object has been signed with an enveloped

14 signature, and the *SigningResponse* is a signed object containing the actual data and the

15 signature. The signing was successful, as indicated by the *Status Type*.

16   The *SigningResponse* includes either the signature document itself or a URL

17 specifying how to obtain the signature document. Specifying a URL is useful when the

18 enveloped signature document is large. In one embodiment of the present invention, the

19 server-side digital signature system also provides storage for documents whose signatures

20 are not needed immediately. Storage of the document is requested by specifying the

21 URL as the value of the *SignatureDocument* attribute of the *SigningResponse* element.

22   The *SigningResponse* element returns either the entire signature document, the

23 signature element, or a URI for the signature document depending on the Return

20

1   *SigningRequest* attribute. The *SignatureResult* element *Return* attribute specifies the type

2   of return, and should always match the request. The present invention passes the

3   signature document as the content of the *SignatureDocument* element within the

4   *SigningResponse*.

5

6   Verification Request

7   Referring again to Fig. 3, the client can also send a Verification Request,

8   requesting either an unsigned verification response or a signed verification receipt, to the

9   server. The server performs verification of a signed object and sends a response back to

10   the client. Documents to be verified may be specified by URL or by uploading the

11   document content from the client to the server. The URL of a document to be verified

12   does not need to match the URL specified in the document signature reference, but the

13   hash of the document to be verified must match the hash of the document reference.

14   When the verification request asks for an unsigned verification as a response, the server's

15   response would be either "Invalid" or "Valid". When the client asks for a signed Receipt,

16   the server produces a signed verification response signed by requesting client.

17   An independent *VerifyInfo* element is used to specify each signature to verify. A

18   single *VerificationRequest* may include one or more *VerifyInfo* elements. All of the

19   *VerifyInfo* elements in each *VerificationRequest* must all be of the same type, either

20   unsigned verification or receipt. Each *VerifyInfo* element must have a unique Id attribute

21   for the request, if more than one is included in the request. Each *VerifyInfo* element must

22   include the signature to be verified and the document information that is necessary for

23   signature information.

1   A *VerificationPolicy* element is used to include options on the degree of

2   credential checking required for verification, whether the *VerificationResult* document is

3   signed at all, by the verifier, and or by the SSDS system.   Options on the format of the

4   *VerificationResult* document, including whether or not a *SSDSSignatureProperties*

5   element is present is also specified in the *VerificationPolicy* element.

6   The signature to be verified can be either the signature element in a detached

7   signature document, or can be the signature from an enveloped signature within an XML

8   document. The signature can be a signature element with the Id attribute, but no content,

9   if the verification server archived the original signature with that Id.

10   The document information that is necessary for signature verification may be one

11   of the following: 1) the actual document contained within a *Document* element; 2) a

12   document identifier contained within a *DocumentIdentifier* element; or 3) a document

13   URL, suitable for fetching the document. If none of this document information is

14   provided, then the document hash contained within the signature is used as the document

15   information. When the actual document is contained within a *Document* element, the

16   data is base64 encoded if the *Algorithm* attribute is specified and has a type

17   corresponding to base64 encoding.

18   Appendix G shows the code format of a *VerificationRequest* using the entire

19   document.   The verification request indicates what type of request is being asked for,

20   either response or receipt, by setting the *Type* attribute to either "response" or "receipt".

21   The XML signature to be verified is included, along with the address of the data object,

22   which is base64 encoded.

1   Appendix H shows an example of code for a *VerificationRequest* using just a

2   signature identifier and document URL. This type of *VerificationRequest* is used when

3   the server archived the original signature with the Signature ID. The signature is verified

4   over the data object specified by the document URL. If the document URL were not

5   specified, the *DocumentHash* element would be used to create the document reference.

6   When the *Document* element content is present, then the signature is verified

7   using the data content in the request. If the *Algorithm* attribute is specified, such as

8   base64 encoding as shown in Appendix G, then the element content is decoded according

9   to the specified algorithm. If the *Algorithm* attribute is not specified, the element content

10   is not specified. If the *Document* element is empty, then a URI attribute is required and is

11   used to fetch the document. If both the *Document* element and the *URI* element are

12   present, then the element content is used to verify the signature and the URI is required to

13   match the reference URI. If the Document element is not present, then the

14   *DocumentHash* element is required and specifies the hash of the document used to create

15   the document reference. The *KeyInfo* element of the signature is used to specify the

16   certificate and the public key used to verify the signature.

17

18   Verification Response

19   Appendix I is an example of code for a *VerificationResponse* according to the

20   present invention is shown. The verification response indicates the result of verifying the

21   signature as requested by the *VerificationRequest* shown in Appendix H. As shown in

22   the code in Appendix I, the *SignatureStatus* was "Invalid" and the *CredentialStatus* is

23

1   "Revoked". The *VerificationResponse* also includes a Server Identifier and a

2   Timestamp.

3   The *VerificationResponse* is an XML element that includes a *VerifyInfo* element

4   corresponding to each *VerifyInfo* element in the *VerificationRequest*. If more than one

5   *VerifyInfo* element is in the response, then each is required to have an Id attribute with the

6   value matching that of the corresponding *VerifyInfo* element in the *VerificationRequest*.

7   The verification response does not include the signature specifications because the

8   requestor obtained the signature specifications when a verification request was made.

9   The *VerifyInfo* element may include a *SignatureReference* element with a *Type*

10   attribute specifying one of the following types of references:

11   1.   The *hash* type has a value containing the encoded hash of the *Signature*

12   element, where the hash that was encoded according the specified algorithm.

13   2.   The *id* type has a value containing the Id attribute of the *Signature*

14   element.

15   3.   The *opaque* type has an opaque value which the Hancock verifier can use

16   to look up the signature.

17   The *SignatureStatus* element in the *VerificationResponse* summarizes the result of

18   signature verification, "Valid", "Invalid", or an error message. Each

19   *VerificationResponse* includes a *Timestamp* element and a *SignatureReference* element.

20   The *Timestamp* element and the *SignatureReference* element are defined the same way

21   that they were for the *SignatureProperties* element. In one embodiment of the present

22   invention, the server-side digital signature system provides signature verification using

24

1 the *SignatureStatus* element. In an alternate embodiment, the server-side digital

2 signature system provides signature verification using the *CredentialStatus* element.

3

4 <u>Verification Receipt</u>

5   A Verification Receipt is a Verification Response that is signed at the server by

6 the Requesting Party. A Verification Receipt allows the Requesting Party to prove to a

7 third party, such as an original signer or an escrow agent, that the Requesting Party

8 performed due diligence by verifying the signature at a specific point in time. The

9 sequence of events for a verification receipt is as follows:

10   1)   Relying party requests signature verification.

11   2)   Relying party receives verification response.

12   3)   Relying party submits verification response for signing.

13   4)   Digital signature signing server produces signed verification response.

14   Appendix J contains code of an example *VerificationResponse* that is a

15 Verification Receipt according to the present invention, as shown by the

16 *"TYPE='receipt'"* field. A *VerificationResponse* that is a Verification Receipt has one or

17 more additional *Signature* elements containing the appropriate signatures. The

18 *SignatureProperties* element in the *Signature* element is used to distinguish the

19 signatures.

20

21 <u>Signature Summary Request</u>

22   A signature summary document is an XML document that provides information

23 about one or more signatures, including information such as the signature algorithm, key

25

1 strength and rating, the availability of associated credentials and credential information,

2 signature property information, and an indication of what portion of the document is

3 included in the signature.

4   Appendices K and L each contain examples of code for a

5 *SignatureSummaryRequest* element. A *SignatureSummaryRequest* element specifies

6 which signatures to summarize, either by specifying a detached signature document or by

7 specifying a document that contains enveloped or enveloping signatures and optionally

8 specifying the signatures within the document. The request includes Document

9 Information and a Signature specification. The Document Information is identical to the

10 Document Identification that is specified for a Verification Request. If a document hash

11 is provided, the server-side digital signature system retrieves the document based on the

12 hash. The Signature specification contains the Ids of the signatures to be summarized. If

13 no signature specification is included, all of the signatures are summarized. This is

14 specified by one or more *SignatureIdentification* elements, with the Id attribute.

15

16 Signature Summary Response

17   The *SignatureSummary* element includes a *SignatureInfo* element for each

18 signature summary. It also always includes a *Disclaimer* element regarding use and

19 warranty. A *SignatureSummary* may be an unsigned response, or a signed receipt, as

20 indicated by the optional TYPE attribute.

21   Appendix M shows example code for a *SignatureSummary*. There is a

22 *SignatureInfo* element for each signature in the signature document. Each *SignatureInfo*

23 element provides information about that signature, including ratings of algorithms, keys

1   and credentials. Ratings are from F- for worst to A+ for best, with +, - or no modifier for

2   each letter. Each algorithm includes the algorithm identifier. Signer names and other

3   information are also provided when available. The document can also include a legal

4   disclaimer, such as "not responsible for use, or warranty".

5   If the *SignatureSummary* contains more than one *SignatureInfo* element, then

6   each *SignatureInfo* element has an Id attribute with the same value as the corresponding

7   *Signature* element. Each *SignatureInfo* element has a *Type* attribute specifying whether

8   the corresponding signature is "detached", "enveloped" or "enveloping". Each

9   *SignatureInfo* includes a *KeySummary* element, a *CredentialSummary* element, and a

10   *SigSummary* element, containing information about the signing key, associated

11   credentials and signature respectively.

12

13   <u>User-Defined Properties</u>

14   End-users can define the properties that they wish to have signed using a modified

15   XML document. The XML content is enclosed in the *UserProperties* element and is

16   included in the signing request. The *UserProperties* element is inside the *Object* element

17   in the *Signature* element. A reference to the *UserProperties* element is included in the

18   *SignedInfo* element, including the user properties in the signature value. In the preferred

19   embodiment, this is used to implement notarization services, to support user CSSD data,

20   and to support other user-defined requirements.

21   The *HancockSignatureProperty* element contains the following components: a

22   Hancock *Timestamp*, a Hancock *ServerIdentifier*, a Hancock *SignatureReference* and a

23   *VerificationURI* . The Hancock *Timestamp* indicates the time of signing at the Hancock

27

1 server, which is provided using the local time of the server. The timestamp is used to

2 sequence events at the server, and is contained within the Timestamp element. The

3 Hancock *ServerIdentifier* is used to identify the particular Hancock server that produced

4 the signature. This component is contained within the *ServerIdentifier* element. The

5 Hancock *SignatureReference* correlates the signature with the log files and other records

6 of a specific Hancock server. This and the Hancock *ServerIdentifier* provide a unique id

7 for the signature, and is contained within the *SignatureReference* element. The

8 *VerificationURI* identifies the default verification server. In an alternate embodiment,

9 this identifier is used to specify an alternate server. The *VerificationURI* is configurable

10 in each Hancock server, and is contained within the *VerificationURI* element.

11 The Hancock *SignatureProperty* element contains an *AuthenticityStamp/License*

12 component that is used to prevent non-authorized users from creating a Hancock

13 *SignatureProperties* element. The *AuthenticityStamp/License* provides verification that

14 the particular digital signing server is licensed. The stamp is created in one embodiment

15 by creating a hash of the Hancock Timestamp, Identifier and Reference String and a

16 confidential phrase.

17

18 Signature Elements

19 Appendix N shows an example of code for a Hancock XML *Signature* element.

20 Items in parentheses are optional. Each *Reference* element contains fields for a *URI*, a

21 *Transform*, a *DigestMethod*, and a *DigestValue*. The *URI* is an address of the data object

22 to be signed. The *Signature* element has an *Object* for each property desired.

28

1   When a signature is detached, it is contained in a document separate from the

2   document that contains the data. In an XML system, when the signature is detached, a

3   signature document contains the XML signature element and a pointer to the second

4   independent document.   The second document contains the signed data content. An

5   enveloped signature is contained in the same document with the data. When the

6   signature is enveloped, a new document is created that contains the XML signature

7   element as well as the data content.   One method of creating an enveloped signature is to

8   add the signature element to the document is in a different namespace from the original

9   document. A second method of creating an enveloped signature is to place the data

10   content as an object inside the signature element.

11   Appendix O is code showing the structure of the *Reference* element of Appendix

12   N in more detail. The *URI* refers to an external document when a detached signature is

13   requested. Hancock uses fully-specified URL's, except when there is an enveloped

14   signature reference or when there is a reference to the Hancock Signature Object. The

15   *URI* refers to an enclosing root XML document for an enveloped signature, and it refers

16   to an XML fragment for a signed signature property. *Transforms, DigestMethod,* and

17   *DigestValue* provide the information needed to obtain the hash of the data content, and

18   the actual hash of the data content.

19   Appendix P is code showing an example of the *KeyInfo* element of Appendix N.

20   In the code shown in Appendix P, a certificate chain is provided that includes an

21   *X508Data* element. The *KeyInfo* element contains key information needed by the verifier

22   to verify a signature. A certificate chain can be provided; in this case, the certificate

23   chain is used to validate the public key associated with the signature. A *KeyName* can

1   also be provided to look up the key at the server. An identifier can also be provided in

2   this field.

3   Appendix Q is code showing the *Object* element of Appendix N in more detail.

4   The *Object* element defines other signature properties. The *Signature* element has an

5   *Object* for each property desired.

6   Appendix R is code for an example of the reference for the

7   *ServerSignatureProperties*. The *HancockSignatureProperties* element is specific to the

8   present invention. Properties to be signed must have a unique ID. In this example, the

9   ID is the *ServerSignatureProperties* ID. The signed info element in the *Signature*

10   element must include a reference for the property. The URI is a fragment for the ID, and

11   in this example, it would be a URI value of #*ServerSignatureProperties*. The

12   *SignatureProperty* element has a unique ID so that it can be referenced from the

13   *Reference* element. This allows the *VerificationResponse* elements to refer to multiple

14   signatures. Hancock generates and assigns an ID to each Signature. The ID is used to

15   match the verification response to the signature.

16   Algorithms are specified using the URI's defined in the XML Digital Signature

17   specification. One embodiment of the present invention uses the RSA-SHA1 algorithm,

18   although other algorithms are well-known in the art.

19   The XML *KeyInfo* element is supported in the Hancock system. Hancock

20   signatures are self-contained and are complete. In one embodiment of the present

21   invention, the Hancock *KeyInfo* element contains the public key X.509 certificate, and in

22   another embodiment, contains the entire certificate chain.

30

1   In the preferred embodiment, the Hancock system has a *X509Data* element with

2   an *X509Certificate* element for each certification in the chain. Each certificate is a

3   base64 encoded X.509 certificate. In an alternate embodiment, the *KeyName* element

4   within the *KeyInfo* element allows the key information to be kept private at the server for

5   validation. The preferred embodiment uses base64 encoding and decoding, SHA-1

6   digest, and RSA with SHA-1 Signature.

7   All elements that are specific to the server-side digital signature system of the

8   present invention, namely the *HancockSignatureProperties* element, the Hancock-

9   specific requests and the Hancock-specific responses, are contained in the Hancock

10   namespace.

11

12   Application Protocol

13   In the preferred embodiment of the present invention, Signing Requests, Signing

14   Responses, Verification Requests and Verification Responses are transported using the

15   SOAP application protocol, allowing remote procedure calls. SOAP defines a wrapper to

16   go around the requests and responses, allowing for routing and processing. One

17   advantage of SOAP, an XML based protocol, is that it is transported over HTTP,

18   allowing access through most firewalls.

19   In other embodiments, the requests and responses according to the present

20   invention are transported by other mechanisms, and can also be archived. For this reason,

21   the request and response names are unique to distinguish them without relying on the

22   transport wrapper. Where XML uses only a *Signing* element, the present invention

23   provides *SigningRequest* and *SigningResponse* elements instead. The *SigningResponse*

31

1 element can be wrapped in a SOAP envelope. The SOAP wrapper can also convey

2 failure information.

3   Appendix S is an example of SOAP code showing a Hancock *SigningRequest*

4 enclosed in a SOAP wrapper. The Internet host is specified as www.signingserver.com,

5 and the content type is specified as XML.

6

7 Integration with Existing PKI Systems

8   According to the present invention, the server-side digital signature system can be

9 integrated with an existing Public Key Infrastructure (PKI) system. In this configuration,

10 the end-entity has a public/private keypair and a public key certificate. The key pair may

11 have been generated using any of a number of possible methods. The private key may be

12 held on the end entity's computer, or it may be in a hardware token. The present

13 invention does not use the private key generated by the existing PKI to generate

14 signatures, but instead uses a private key it creates on behalf of a signer to generate

15 signatures for that signer.

16   The server-side digital signature system of present invention can load an

17 externally generated private key via its core encryption engine. In alternate

18 embodiments, this mode of operation is used to support various signing scenarios. The

19 PKI generated private key can optionally be used to authenticate to the server-side

20 signature generator for purposes of signing, verification, or administration.

21   The PKI-generated private key can also be used to support two-factor content.

22 Two-factor content provides an additional level of non-repudiation support for any

1 server-generated signature utilizing it. In one embodiment of the present invention,

2 private keys that have not been generated by the server-signer can be used for signatures.

3 Alternate embodiments provide a number of methods for end-entity private

4 signing key management. One method is to upload the end-entity private key to the

5 server. A second method is to use only signatures generated using some form of two-

6 factor content. According to a third method, the server provides some low-level

7 interfaces to the client, such as a hash function or a function to create an XML signature

8 document given a signature and a key.

9 While the invention has been described in a particular XML implementation, it

10 will be apparent to those skilled in the art that many other implementations are possible.

11